Esercizio di Sincronizzazione tra Processi:
Ponte a Senso Unico Alternato con Capacità Limitata e Senza Starvation

Supponiamo sempre di avere un ponte stretto che permette il passaggio delle auto solo in un verso per volta, a senso unico alternato. Supponiamo che la resistenza del ponte sia limitata, e quindi la sua capacità di carico sia limitata ad un numero massimo di auto C contemporaneamente presenti sul ponte. Si vogliono evitare situazioni di starvation, e perciò si deve provvedere a limitare ad M il numero di auto che possono consecutivamente passare in una direzione.

1° Soluzione con l'uso del costrutto monitor

{soluzione che fa uso di due code diverse, una per il problema della direzione e del numero di passaggi, l'altra per il problema della capacità }

```
program PonteASensoUnicoConCapacitaLimitataETransitoAlternato;

const M = ... ; CAPAC = ...;
type dir  = ( su, giu );

{Notiamo che la struttura dei processi auto non cambia rispetto a quella del ponte semplice }
type auto ( d : dir ) = process ;
begin   Ponte. IN (d);
        < transita >
        Ponte .OUT (d);
end;
```

{Con l'introduzione del vincolo della capacità del ponte ed anche del numero massimo di passaggi in una direzione determinata in caso di richieste nella direzione opposta, si devono introdurre nel monitor:

- due variabili condition per l'attesa di accesso al ponte - attesa_dir [dir] - in cui sono sospese le auto quando il ponte non ne permette l'accesso a causa della direzione o del numero di passaggi. Questa volta servono due variabili condition perché a causa della limitazione al numero massimo di passaggi ci potrebbero essere auto sospese contemporaneamente in entrambe le direzioni,
- una coda di sospensione per raggiunta capacita' massima del ponte : attesa_cap; per questa invece basta una sola direzione
- contatori opportuni, di auto in passaggio sul ponte: nautoponte, numero totale di auto sul ponte, npassaggi, auto già transitate consecutivamente in una direzione, nauto, di auto che hanno gia' eseguito la in e non hanno ancora fatto la out, nauto.

La soluzione prevede di massimizzare l'utilizzo della risorsa:
- in caso di forte traffico nei due sensi, il ponte è gestito a senso unico alternato;
- in caso che ci sia traffico in un solo verso, il verso corrente non viene bloccato, ma passa a tranche di M auto;
- in caso di passaggio di un numero di auto inferiore a M in una direzione, e di traffico nell'altra, quando sono terminate le prime, si dà il via alle auto nella direzione opposta anche se non si e' arrivati ad un numero di passaggi pari ad M.

```
type bridge = monitor ;

var  direz : dir ; nauto, nautoponte, npassaggi : integer;
     attesa_cap : condition;
     attesa_dir: array [dir] of condition;
```

```
procedure entry IN ( d : dir);
begin
    if (d <> direz and nauto <> 0 )   or
      ( d = direz and npassaggi >= M and attesa_dir[other(d)].queue)
         then  attesa_dir[d]. wait ;
         { c'è sospensione se la direzione non è quella corretta e anche se sono già passate M auto
         per la direzione corrente con auto in attesa su quella opposta}
    nauto := nauto +1;
    direz := d ; npassaggi := npassaggi +1;
    {in questo momento non mi considero ancora sul ponte devo prima controllare la capacità}
    if nautoponte = CAPAC then attesa_cap.wait;
    nautoponte := nautoponte + 1;
end;

procedure entry OUT (d : dir ) ,
begin
    nautoponte := nautoponte - 1 ;
    { segnalazione delle auto eventualmente sospese per vincoli di capacità del ponte }
    attesa_cap.signal;
    { la signal non ha effetto se la coda è vuota }
    nauto := nauto - 1;

    { segnalazione delle auto nella direzione opposta, se ce ne sono, o di auto nella stessa
         direzione, consentendo ancora M passaggi }
    if ( nauto = 0 ) then
    begin
         npassaggi := 0;
         if ( attesa_dir [ other (d) ]. queue  then
            while attesa_dir[other(d)].queue and npassaggi < M do
                 attesa_dir [other(d) ].signal;

{  le segnalazioni vengono fatte solo fino a quando si e' sicuri che non sono possibili risospensioni del
segnalato nella stessa coda:  questa eventualità causerebbe una situazione di stallo: il segnalante
segnala, il segnalato si risospende, il segnalante segnala, etc. }

         else if attesa_dir[d].queue then
                 while attesa_dir[d].queue and npassaggi < M do attesa_dir [d].signal
    end;

end;

var  Ponte : bridge;
    auto1, auto2,.... : auto(su);   auto100, auto101,.... : auto(giu);

begin end.
```

2° Soluzione con l'uso del costrutto monitor

{soluzione che fa uso dello stesso tipo di coda diverse (ma comunque una per ogni direzione) per il problema della direzione e del numero di passaggi e per il problema della capacità. In effetti la prima soluzione di prima è un po' ridondante }

```
program PonteASensoUnicoConCapacitaLimitataETransitoAlternato;

const M = ... ; CAPAC = ...;
type dir  = ( su, giu );

{la struttura dei processi auto non cambia rispetto alla prima soluzione }
type auto ( d : dir ) = process ;
begin   Ponte. IN (d);
        < transita >
        Ponte.OUT (d);
end;
```

{Questa volta usiamo come variabili del monitor:

- due variabili condition per l'attesa di accesso al ponte - attesa_dir [dir] - in cui sono sospese le auto quando il ponte non ne permette l'accesso a causa della direzione o del numero di passaggi o della capacità. Questa volta servono due variabili condition perché a causa della limitazione al numero massimo di passaggi ci potrebbero essere auto sospese contemporaneamente in entrambe le direzioni,
- contatori nauto, numero totale di auto sul ponte ed npassaggi, auto già transitate consecutivamente in una direzione.

}

```
type bridge = monitor ;

var     direz : dir ;
        nauto, npassaggi : integer;
        attesa_dir: array [dir] of condition;

procedure entry IN ( d : dir);
begin
    if (d <> direz and nauto <> 0 )   or (nauto+1>CAPAC) or
      ( d = direz and npassaggi >= M and attesa_dir[other(d)].queue )
         then  attesa_dir[d]. wait ;
```

{ c'e' sospensione se la direzione non e' quella corretta e anche se sono già passate M auto per la configurazione corrente con auto in attesa sull'altra oppure si superasse la capacità massima}

```
    nauto := nauto +1;
    direz := d ; npassaggi := npassaggi +1;
end;
```

```
procedure entry OUT (d : dir )
begin
        nauto := nauto - 1;
        if attesa_dir[d].queue and (npassaggi < M or not attesa_dir[other(d)].queue)
        then attesa_dir[d].signal;
        else if ( nauto = 0 ) then
                begin
                npassaggi := 0;
                while (attesa_dir[other(d).queue] and
                        (npassaggi < M or not attesa_dir[d].queue) and
                        nauto < CAPAC) do
                  attesa_dir[other(d)].signal;
end;

var  Ponte : bridge;
    auto1, auto2,.... : auto(su);   auto100, auto101,.... : auto(giu);

begin end.
```

Con l'uso delle regioni critiche condizionali:

```
{una versione semplificata rispetto al monitor che non riesce a garantire stretta alternanza}
type d = (su,giu);

VAR ponte : shared record
    cont : integer;        { contatore degli utenti che hanno acquisito il ponte }
    dir  :  d;             { direzione corrente del ponte}
    npass:integer;         { contatore dei passaggi nel verso corrente}
                end record;

procedure IN (miadir : d);
region ponte
when ((((miadir = dir) and (npass < M)) or  ((miadir <> dir) and (cont = 0)
{si entra se o nella direzione corretta e entro M nella direzione opposta, ma non ci sono auto }
do
  if dir <> miadir then npass := 0;
   dir := miadir; cont + := 1;
   npass + := 1;
end;
end IN;

procedure OUT (miadir : d);
region ponte
do
   cont - := 1;
   if (cont = 0) and ( npass = M) then npass := 0;
end region;
end OUT;
```

{Se il ponte e' vuoto, il primo che arriva puo' sempre accedere al ponte indipendentemente dal verso corrente, che viene imposto dal processo. Infatti con le regioni critiche io non ho la possibiltà di controllare il risveglio in modo efficace
In caso di traffico, le richieste sono servite fino al massimo M, dopo si puo' cambiare la direzione. Come gia' detto, la soluzione non garantisce l'alternanza. }

```
process type utente;
var dir : d;
begin   loop   <scegli dir>
                  IN (dir);
                  <passa sul ponte>
                  OUT (dir);
          end loop;
end utente;

var u1, ..., un : utente;
begin    region ponte do    npass := 0; cont := 0; dir := su;  end region; end;
```

Con l'uso di ADA

{Anche in ADA, possiamo scrivere una versione semplificata dell'algoritmo implementata dal monitor, come segue. (Per una traduzione piu' precisa si veda l'esercizio seguente).}

```
package PonteETuttoIlResto is

task ponte is
  entry insu;
  entry ingiu;
  entry outauto;
end ponte;
task body ponte is

    cont : integer;
                { contatore degli utenti che hanno acquisito il ponte }
    dir  :  d;
                { direzione corrente del ponte}
    npass:integer;
                { contatore dei passaggi nel verso corrente}
begin
 cont := 0; dir := su; npass :=0;
 loop
    select
        when (((dir = su) and (npass < PASSAGGI)) or ((dir <> su) and (cont = 0)))
        accept insu do
                cont + := 1;
                if dir <> su then dir := nord; npass := 0; end if;
                npass + := 1;
        end insu;
        or
        when (((dir = giu) and (npass < PASSAGGI)) or ((dir <> giu) and (cont = 0)
        accept ingiu do
                cont + := 1;
                if dir <> giu then dir := giu; npass := 0; end if;
                        npass + := 1;
        end ingiu;
        or
        accept outauto do
                cont - := 1;
                capac  - := miopeso;
         end outauto;
    end select;
end loop;
end ponte;
```

{Il non-determinismo di ADA con questa soluzione non consente una alternanza stretta dei due versi: infatti, il costrutto select non consente di ottenere direttamente l'alternanza. Studiare una soluzione alternativa.}

```
task type utenteG is end utenteG;
task type utenteS is end utenteS;
task body utenteS is
begin
  ponte.insu;
   < passa sul ponte >
  ponte.outauto;
end utenteS;

task body utenteG is
begin
  ponte.ingiu;
   < passa sul ponte >
  ponte.outauto;
end utenteG;

un1, ..., unn: utenteS;
us1, ..., usm: utenteG;
begin
end PonteETuttoIlResto.
```